CS 345: Homework 3 (Due: Oct 14, 2013 before 10:01am)

Rules. This is to be completed no later than the start of class on the day it is due. Earlier submission is also possible by email (read Handout 1).

Notation that will be enforced: Know your bits and Bytes. 1MB is usually 1,000,000B if it refers to hard disk space; if you are not convinced, buy a hard disk drive and read the warranty information. For main memory it is equal to 1MiB (mebi-byte, an SI notation). 1MiB is 1,048,576B. Nowadays we use 1TB for disk space i.e. 10^{12}B. The moment you format the hard disk and install a file system (or the PC manufacturer does so for you) capacity gets stated by the OS in `TiB` or `Gib` even if they call it "TB" or "GB". Note that during formatting capacity goes down negligibly 1%. But a 1TB disk drive, will show up on a computer as having space 0.91 "TB" i.e. 0.91TiB since $10^{12}/2^{40} \approx 0.91$. If you express the 0.91TiB in GiB it is not 910GiB but 0.91*1024GiB or roughly 931GiB. And a capital `B` is a byte. A lower case `b` is nonsense; use `bit` for a bit. We are going to use in the remainder the SI system (aka System Internationale). There we have `kibi, mebi, gibi, tebi` in SI, but all characters other than the i in KiB,MiB,GiB,TiB are in capital case. And do not forget k is the thousand multiple as in $1kgr = 1000gr$.

```
 Hard disk drive capacity   SI system notation
1KB = 1000B                 1KiB (kibi-byte) =  1024B = 2**10
1MB = 1,000,000B            1MiB (mebi-byte) =  1024*1024B = 1048576B = 2**20
1GB = 1,000,000,000B        1GiB (gibi-byte) =  2**30
1TB = 1,000,000,000,000B    1TiB (tebi-byte) =  2**40
1TB is 10**12 / 2**40 TiB i.e.  1TB = 0.91 TiB
1TiB = 1024 GiB                 1TB = 0.91 * 1024 GiB ~ 931 GiB
```

Problem 1. (45 points) **Paper by Brin and Page**
Read the paper that describes Google as it was originally designed and answer some questions on the paper (section C5, link L1, has the paper in URL and also a local copy of the paper in pdf `P1.GoogleBrinPage.pdf`). They have some differences so read BOTH of them. If an answer is not in one it might in the other!
Subject 3 and a bit of Subject 4 might also prove useful as far as hash tables are concerned, if you are not familiar with it from an elementary data structure course such as CS 114 or CS 435. Link
`http://en.wikipedia.org/wiki/Hash_table`
might also help.
Answer the following questions after reading the paper. The answers you provide are and should be drawn from the paper and thus justification will be to paper-available information.

(a) According to the paper, what was the size (bytes) of Google's Lexicon around 1998?

(b) According to the paper how many bits for a `docID`? Quote the paper.

(c) Describe the data structures used by Google for indexing only (not all of them are listed in the architecture figure).

(d) Does Google (1998) use a hash table for the dictionary? What do they use? Why ?

(e) How many bytes are assigned to each hit (of the hitlist)? How many types of hits? What are they (types of hits)?

Problem 2. (45 POINTS) **Hash Table design ala Google/1998**
You have 1GiB of main memory of which 124MiB are being used by the operating system plus related programs such as those manipulating tables A and T below.

You are asked to organize the additional space to support a hash table along the lines of the paper where words are stored in a contiguous table which is an array A of characters delimited by a null character `\0`. A hash table T will store a `wordID` along with a reference (pointer or index) p to A. That way a `wordID` will be associated with a specific `word`.

The average length of a `word` is given as 7 ASCII characters (1 ASCII is 7 bits but occupies one byte). A `wordID` and p can only be in multiples of one byte (i.e. 1B, 2B, 3B but and thus 23bits won't be an option) for efficiency. Organize T and A for maximum efficiency.

In an efficient implementation

- (a) How many words n can the scheme support,
- (a) How big would the hash table size m (number of entries) be,
- (c) How many bits for a `wordID`,
- (d) How many bits for pointer p
- (e) How much space (bytes) will you scheme use for T,
- (f) How much space (bytes) will you scheme use for A,
- (g) What is the total space $A + T$ used by your scheme?

Justify your answers and choices. Round to nearest million for n, m, T, A but make sure you don't exceed the amount of available memory (i.e. $A + T$ should be accommodated easily by the available memory). Make sure that you fill the following table with data.

n	=
m	=
wordID	=
p	=
T	=
A	=
A+T	=